Constrained Social Community Recommendation

Xingyi Zhang*
The Chinese University of Hong Kong
Hong Kong SAR, China
xyzhang@se.cuhk.edu.hk

Shuliang Xu
Wenqing Lin†
Tencent
Shenzhen, China
shuliangxu,edwlin@tencent.com

Sibo Wang†
The Chinese University of Hong Kong
Hong Kong SAR, China
swang@se.cuhk.edu.hk

ABSTRACT

In online social networks, users with similar interests tend to come together, forming social communities. Nowadays, user-defined communities become a prominent part of online social platforms as people who have joined such communities tend to be more active in social networks. Therefore, recommending explicit communities to users provides great potential to advance online services.

In this paper, we focus on the constrained social community recommendation problem in real applications, where each user can only join at most one community. Previous attempts at community recommendation mostly adopt collaborative filtering approaches or random walk-based approaches, while ignoring social relationships between users as well as the local structure of each community. Therefore, they only derive an extremely sparse affinity matrix, which degrades the model performances. To tackle this issue, we propose ComRec which simultaneously captures both global and local information on the extended graph during pre-computation, speeding up the training process on real-world large graphs. In addition, we present a labeling component to improve the expressiveness of our framework. We conduct experiments on three Tencent mobile games to evaluate our proposed method. Extensive experimental results show that our ComRec consistently outperforms other competitors by up to 12.80% and 6.61% in the corresponding evaluation metrics of offline and online experiments, respectively.

CCS CONCEPTS

• **Information systems** → **Social recommendation**; **Social networks**; **Recommender systems**; • **Computing methodologies** → **Learning latent representations**.

KEYWORDS

Constrained Community Recommendation; Social Network; Graph Neural Networks

ACM Reference Format:
Xingyi Zhang, Shuliang Xu, Wenqing Lin, and Sibo Wang. 2023. Constrained Social Community Recommendation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3580305.3599793

*This work was done when the first author did an internship at Tencent.
†Corresponding author.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

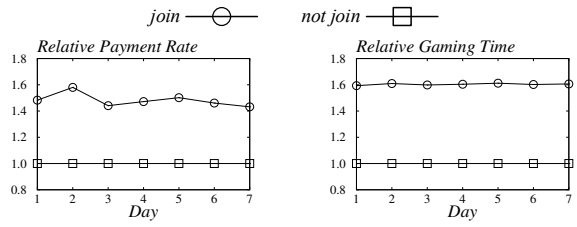
ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599793

1 INTRODUCTION

Communities exist in droves in online social networks, as the tendency of people with similar interests to group together is inherent in social networks [2]. On the one hand, users in online communities benefit from sharing information with each other. Meanwhile, establishing connections with other members, in turn, facilitates online services. For instance, Figure 1 shows the statistics of players in Tencent mobile game X over a consecutive period of 7 days. We can observe that the players that have joined a club tend to be more active. Compared with those that have not joined any in-game club, the relative improvement of the gaming time (resp. payment rate) of players that have joined a club is 60.4% (resp. 48.2%) on average. On the other hand, with the mushrooming communities in online social networks, users have difficulties in choosing the appropriate community to join. Therefore, practitioners, as well as researchers, have sought to devise approaches [3] to achieve community growth in online social networks.

In this paper, we focus on the problem of recommending communities for users to join in social networks [32], where a community is an explicit social unit consisting of users. In particular, only user interactions and community memberships are given in our settings. Different from item recommendation scenarios where users can select more than one item or conventional community recommendation scenarios where users may belong to multiple communities, we have a further constraint that each user can only join at most one community at any time. This constraint does exist in many real-world applications. For example, in many online games, each player can only join at most one in-game club. Other real-world scenarios include *(i)* paper submission platform can recommend journals/conferences for researchers. Each time, researchers can only submit their article to one publication; *(ii)* recruitment websites can provide job seekers with valuable opportunities to find jobs that match their interests. However, one can only accept one offer at a time; *(iii)* mobile operator companies can provide personalized recommendations for different customers. Each time, customers can only choose one plan for each individual phone number. Compared with previous recommendation problems, this constraint severely restricts the number of positive training samples, making this problem rather challenging.

Earlier community recommendation attempts to adopt similarity search strategies. For instance, Spertus et al. [32] calculate different similarities, such as cosine distance and inverse document frequency, to recommend communities for users. Subsequently, as



(a) Payment rate (b) Gaming time

Figure 1: Relative improvements of two important statistics in Tencent game X. The statistics of players that have not joined any club are normalized to 1.

collaborative filtering (CF) plays a significant role in recommendation systems, many research studies [6, 30, 36, 44] fall into this category. These approaches are mainly evaluated on Orkut and Youtube networks, where users may belong to multiple communities. In addition, most of these approaches require side information such as user-item interactions or rating history and further assume that each user has joined at least one community, which is over-idealized in real scenarios.

Recently, the triumph of *graph neural networks (GNNs)* has attracted researchers to design GNN-based approaches for recommendation tasks, such as item recommendation [12, 45, 48, 56] and group recommendation [5, 53], which mainly focus on the user-item bipartite graph while ignoring abundant social interactions between users. It is proven that these social connections facilitate recommendation systems [35] as the choices of users can be highly likely influenced by others around them [25]. Consequently, recent social recommendation approaches [8, 49, 50] further consider social relationships between users. Nonetheless, complex attention operations make them hard to train on large graphs, while a daily update is usually required in real online applications [60]. Meanwhile, a few recent studies [46, 51, 59] evaluate GNN expressiveness, achieving superb performance on link prediction and classification tasks. Accordingly, designing expressive GNNs that can distinguish community/subgraph structures is a promising direction in recommendation systems. Besides, previous GNN-based frameworks only capture either local information by shallow architectures or global information by stacking multiple layers, while both should be adequately captured in the community recommendation problem.

Contribution. Motivated by the above observations, we present *ComRec*[1], an expressive and effective GNN-based framework for constrained social community recommendation problem. Our proposed framework consists of three key components, labeling mechanism, global propagation, and local propagation. The first component is a labeling mechanism that aims to empower the expressiveness of ComRec, which is inspired by recent subgraph representation learning approaches [46, 59]. It has been empirically proved [1, 46] that the ability to differentiate nodes inside and outside subgraphs helps GNNs generate different embeddings of non-isomorphic subgraphs, which may be undistinguishable using a standard message-passing GNN [9]. In particular, we assign additional labels to nodes according to their relations with communities, enhancing the expressiveness of the hidden representations of nodes during the feature propagation process. Besides, we prove that ComRec is more expressive compared with conventional message-passing GNNs [9, 11, 56].

[1]Constrained Social Community Recommendation.

Table 1: Frequently used notations.

Notations	Descriptions
$\mathcal{G}, \mathcal{G}^u, \mathcal{G}^{uc}$	Extended graph, user-user social graph, and user-community bipartite graph.
$\mathcal{U}, C$	The set of users and communities.
n, n_c, F	The number of users, communities, and features.
$\boldsymbol{A}, \boldsymbol{D}, \boldsymbol{X}$	Adjacent, degree, and feature matrix.
$\boldsymbol{H}, \boldsymbol{Z}$	The hidden representation matrix and the output embedding matrix.
$\mathcal{N}(v)$	The set of out-neighbors of node v in $\mathcal{G}$.
$\mathcal{N}_c(v)$	The set of out-neighbors of node v in the subgraph generated by community c and its members.
α, ϵ	The stopping probability and threshold.

In real-world recommendation applications, a daily update is usually required [60], posing a challenge to model scalability as well as training efficiency. To tackle this issue, we decouple the feature propagation process from non-linear transformation operations, following existing popular solutions like PPRGo [4]. In PPRGo, it shows the potential ability of *personalized PageRank (PPR)* [26] to incorporate adaptive multi-hop information of different nodes without the explicit message-passing scheme. However, PPRGo still requires linear running costs for the feature propagation. In particular, it requires $O(\frac{(n+n_c)(\bar{d}+F)}{\epsilon})$ propagation time complexity, where $\bar{d}$ is the average node degree, ϵ is the threshold parameter to balance the computation efficiency and the approximation quality. Such a running cost is still high for our demands as we regularly update our model on a daily basis. In our second component, we present a column-wised propagation strategy so that we propagate features one by one after a column normalization. By such a strategy, we can significantly reduce the running cost to $O(\frac{F\sqrt{n+n_c}}{\epsilon})$, which is sub-linear and super efficient. In particular, our ComRec can finish the propagation process on graphs containing over 20 million nodes using a CPU-only machine in 30 seconds while still achieving superb recommendation results as we will show in our experiments.

Notice that most of the previous approaches only consider incorporating information within the neighborhood of the same hop for each node while ignoring the local structures of different node types. Due to the explicit constraint, interactions between users and communities are exceedingly sparse in our setting. To further delineate the personal influence of community members as well as make full use of the local structural information, we adopt the third component that aggregates member interactions via local propagation to generate more informative representations of communities.

In our experiment, we compare ComRec against other recommendation approaches on the constrained community recommendation task. We conduct experiments on three Tencent mobile games with four real graphs. In addition, we deploy the proposed ComRec in one Tencent mobile game with a real recommendation application. Extensive experiments show that our ComRec consistently outperforms other competitors on all datasets. Our contributions can be summarized as follows:

- We formulate a newly defined recommendation problem, constrained social community recommendation;
- We propose ComRec, an expressive, efficient, and effective framework with three key components for the constrained social community recommendation task;
- We conduct extensive offline experiments on real-world large graphs and shows that ComRec takes the lead by up to 12.80% and 6.61% in the corresponding metrics, respectively.
- We deploy the proposed approach in one Tencent game with a real application. Extensive online experiments show that ComRec takes the lead by at least 2.69x times in terms of running time and up to 29.09% in terms of the corresponding metrics.

2 PRELIMINARIES

2.1 Problem Definition

Let $\mathcal{U} = \{u_1, ..., u_n\}$ and $C = \{c_1, ..., c_{n_c}\}$ denote the sets of users and communities, respectively, where n is the number of users and n_c is the number of communities. We use $\mathcal{G}^{uc} = (\mathcal{U}, C, \mathcal{E}^{uc})$ to denote an undirected user-community bipartite graph, where an edge $\langle u_i, c_j \rangle \in \mathcal{E}^{uc}$ indicates user u_i has joined in community c_j. In addition, we assume that social relationships between users are known in advance, and we use $\mathcal{G}^u = (\mathcal{U}, \mathcal{E}^u)$ to denote the undirected user-user social network. Given the *extended graph* $\mathcal{G} = \mathcal{G}^u \cup \mathcal{G}^{uc}$ and feature matrix $\boldsymbol{X} = concat_{col}(\boldsymbol{X^c}, \boldsymbol{X^u}) \in \mathbb{R}^{(n+n_c)\times F}$, where $\boldsymbol{X^c} \in \mathbb{R}^{n_c \times F}$ denotes community features and $\boldsymbol{X^u} \in \mathbb{R}^{n \times F}$ denotes user features, respectively, and $concat_{col}$ is the column concatenation function, social community recommendation is the task to recommend communities to users who have not joined any community yet. Besides, we further have a *constraint* that each user can only join at most one community at the same time, which is typical in real applications. For example, in many online games, each player can join at most one club. Table 1 lists the notations frequently used in this paper.

Definition 1 (Constrained Social Community Recommendation). *Let $\mathcal{G} = \mathcal{G}^u \cup \mathcal{G}^{uc}$ be an undirected graph, $\boldsymbol{X^u}$ and $\boldsymbol{X^c}$ be feature matrices. In addition, each user can only join at most one community. The goal of the constrained social community recommendation task is to recommend communities for users who have not joined a community yet under the above constraint.*

Remark. Notice that in graph mining tasks like community detection, the goal is to discover groups of nodes that are more similar or densely linked to each other than to the other nodes in complex networks. The community is implicit in such scenarios. However, in community recommendation, the community is explicit and created by users in the networks. For example, in Tencent mobile game X, a player can create a club and others can join an existing club. In this paper, we use a *community node* to denote an existing community and use *community* to denote the subgraph generated by a community node and its members.

2.2 Related Work

To the best of our knowledge, our work is the first to study the constrained social community recommendation problem. Next, we briefly review several closely related recommendation approaches.

Collaborative filtering in community recommendation. Earlier attempts mainly employ CF approaches for the community recommendation problem. CCF [6] considers both community-user and community-description co-occurrences to compute the joint probability distribution over the community, user, and description. Pairwise PLSI [30] employs the pair-wise learning approach to maximize the differences in user preference between two given communities according to their historical records. Wang et al. [44] incorporate the implicit user-community rating to generate temporal user-community affinity. Recently, neural CF approaches, e.g., [12, 45, 48], have shown the effectiveness of GNN in recommendation systems. Each community can be treated as a virtual user in the graph $\mathcal{G}$ and thus GNN-based CF approaches can also be adapted to this problem. Most of these approaches assume that each user in the graph will join at least one community. Nevertheless, this assumption becomes over-idealized in real-world applications. For example, in Tencent game X, only around 10% of the users have joined an in-game community. Furthermore, due to the explicit constraint, the interactions between users and communities are extremely sparse than the conventional community/item recommendation problems.

Social Recommendation. Employing social relationships is another choice to facilitate recommendation systems. RecSSN [34] captures the global information of each user via the relation network. Later approaches mainly adopt GNNs for the social recommendation task. For instance, GraphRec [8] learns user representations by aggregating features from two different perspectives. DiffNet++ [49] presents social diffusion and interest diffusion in a unified framework to aggregate embeddings from different aspects. FeSoG [23] further addresses the privacy protection challenge by employing the pseudo-labeling technique. These models mainly adopt complex attention mechanisms and thus are hard to scale to real-world million-node graphs.

Group Recommendation. Group recommendation is the task to make recommendations to a group of users. The key challenge in this task is how to infer the decision of a group of users. SIGR [54] adopts attention mechanisms to learn the social influence of users in different groups. GroupIM [29] maximizes the mutual information between representations of groups and members to aggregate user preferences. If we treat groups as virtual users in the graph, group recommendation can be considered as the reverse problem of community recommendation. However, since the interaction data between users and groups is extremely sparse, existing group recommendation approaches require additional side information, e.g., user-item interactions, to improve the estimation accuracy of user preferences in groups. Therefore, how to adapt group recommendation approaches to the community recommendation problem without side information is still an open problem.

3 COMREC FRAMEWORK

Recap from Section 2.1 that in constraint social community recommendation task, each user can only join at most one community at the same time. There are roughly two major challenges in this problem. Firstly, rich interactions between users and items/communities are available in most recommendation scenarios, e.g., [6, 12, 13, 30, 44, 45, 49]. However, due to the explicit constraint in our setting, each user can only have at most one positive sample. Furthermore,

the majority of users may not have joined any community yet, resulting in extremely sparse positive training data. Secondly, no additional item information is given. Therefore, recommending communities for users to join belongs to the cold-start problem, which makes it rather challenging. These challenges call for a more expressive framework to better capture the inherent information from different perspectives of graph structures. Section 3.1 briefly illustrates the ComRec framework. Details of the labeling mechanism are provided in Section 3.2. Section 3.3 elaborates on the global propagation process. Section 3.4 proposes the local propagation process. Section 3.5 introduces the prediction process.

3.1 Overview

Figure 2 shows the architecture of our proposed ComRec model, which includes three key components, i.e., labeling mechanism, global propagation, and local propagation.

The first component is the labeling mechanism on the extended graph $\mathcal{G}$, which is generated by the user-user social network $\mathcal{G}^u$ and the user-community bipartite graph $\mathcal{G}^{uc}$. Since users are influenced by both neighboring user nodes and the community node, combining the social network with the user-community bipartite graph provides opportunities to learn more informative representations from two different perspectives. In addition, each community can be regarded as a subgraph in the extended graph. Since the ability to distinguish different subgraphs is important for generating more informative representations, we incorporate a labeling strategy to enhance the expressiveness of our ComRec.

The second component is global propagation which decouples the feature propagation from the complicated parameter training process. Different from previous attempts in social recommendation systems, e.g., [8, 49, 50], which jointly train embeddings from graphs $\mathcal{G}^u$ and $\mathcal{G}^{uc}$ and then concatenate the embeddings together to obtain the final prediction, we directly propagate information on the extended graph, which retains rich user-user interactions for generating the embeddings of the community nodes. To speed up the training process on large graphs with tens of millions of users, we design an efficient global propagation algorithm that only incurs sub-linear time complexity.

The third component is local propagation. Notice that in previous recommendation approaches, node features are mostly propagated within the neighborhood of the same hop for each node without distinguishing different structures or node types. Besides, the positive training data is extremely sparse in our setting. The local feature propagation on subgraphs further captures the structures of each community, aiming to generate more informative representations.

3.2 Labeling Mechanism

Graph Neural Networks (GNNs) have achieved great success in the past few years. Most of the proposed GNNs follow the framework of *message-passing graph neural network (MPGNN)* [9], which updates the representation of a node by iteratively aggregating information from its neighbors. The representation of node v in the k-th standard MPGNN layer, denoted as $\boldsymbol{h}_v^{(k)}$, can be formulated as follows:

$$\boldsymbol{a}_v^{(k)} = \text{AGGREGATE}(\{\boldsymbol{h}_u^{(k-1)} : u \in \mathcal{N}(v)\})$$
$$\boldsymbol{h}_v^{(k)} = \text{COMBINE}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)})$$

where $\mathcal{N}(v)$ is the neighbors of node v in graph $\mathcal{G}$.

As discussed in [1, 46], differentiating nodes inside and outside subgraphs enables GNNs to generate different embeddings of non-isomorphic subgraphs. We assign each node a label vector to indicate the affiliation information. Given a node x and a community c, the label of node x is defined as follows:

$$\boldsymbol{l}_x(c) = \begin{cases} 1, & \text{if } x \in c \\ 0, & \text{if } x \notin c \end{cases}$$

The node x can be either a user node or a community node. By introducing additional labels to MPGNN, it enhances the representations of nodes in graph $\mathcal{G}$. The following theorem shows that the labeling mechanism improves the expressive power of MPGNN.

Theorem 1. *Labeling mechanism enhanced message-passing framework improves the expressiveness of the MPGNN.*

Proof. Firstly, we prove that if a standard MPGNN can distinguish two different subgraphs, then the labeling mechanism enhanced framework can also generate different embeddings for them. For the labeling mechanism enhanced message-passing framework, the representation of the node u at the k-th layer is:

$$\hat{\boldsymbol{h}}_v^{(k-1)} = \text{CONCAT}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{l}_v).$$

It is easy to verify that there always exists a mapping ϕ such that

$$\phi(\hat{\boldsymbol{h}}_v^{(k-1)}) = \boldsymbol{h}_v^{(k-1)}.$$

Therefore, given a standard MPGNN framework $\mathcal{M}$, there always exists a labeling mechanism enhanced message-passing framework that can produce the same embedding as $\mathcal{M}$. If $\mathcal{M}$ can distinguish two non-isomorphic subgraphs, then the labeling mechanism enhanced message-passing framework can also generate different embeddings for these two subgraphs.

Secondly, we need to show that there exist pair of non-isomorphic graphs that MPGNN cannot distinguish while labeling mechanism enhanced message-passing framework can. Consider two non-isomorphic graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ in Fig. 3, where c_1 and c_2 denote community nodes. Subgraphs with blue nodes and edges between them indicate a community. No additional feature information is given. For a standard MPGNN, the representations of node u and node v will be the same since the computational subtrees generated by their neighborhood are homogeneous, see Fig. 3(b). The representations of other nodes will also be the same if we simply use an MPGNN to aggregate information. However, if additional label information is incorporated into the message-passing framework, it can distinguish u from v because u has three in-community neighbors while v has three neighbors outside the community. □

Relaxation. Even though the labeling mechanism can improve the expressive power of MPGNN, it requires additional cost since we further consider a one-hot label vector. If we utilize the batch training strategy, i.e., we jointly incorporate labels to a batch of nodes and then propagate information among edges, the labels may become inconsistent in different batches. To tackle this issue, we only keep a one-dimensional label for each node. In specific, if a node belongs to one community, then it will have the label 1; otherwise, it will have the label 0. Notice that Theorem 1 still holds after relaxation. In addition, as discussed in [46], adding labels

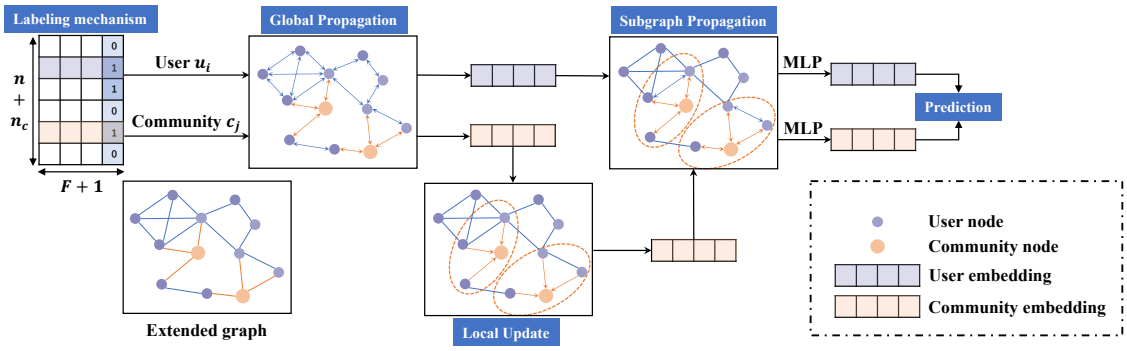


Figure 2: The overall framework of ComRec, which consists of three key components: labeling mechanism, global propagation, and local propagation. The arrows show the direction of feature propagation. Each dashed area in the local update part and the subgraph propagation part is a subgraph consisting of nodes within a community.

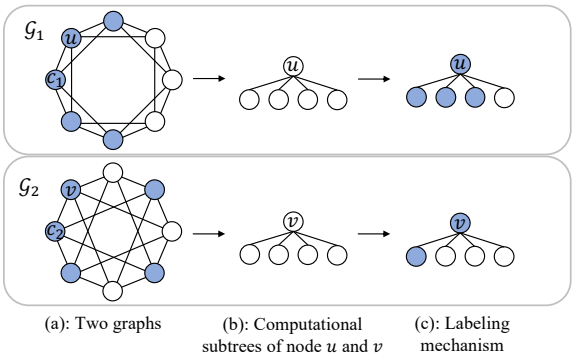


Figure 3: Two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ with communities (blue) generated by communities c_1, c_2, and their members, respectively. The computational subtrees of node u and v are different in standard MPGNN and MPGNN with the labeling mechanism.

cannot avoid inconsistencies between batches and the relaxation here is a trade-off between efficiency and expressiveness.

3.3 Global Feature Propagation

According to the organization principle of social networks [25], the preferences of users are highly likely influenced by their directly connected nodes. The global propagation component aims to incorporate multi-hop information into the representations of each node from the extended graph $\mathcal{G}$ with additional social information from $\mathcal{G}^u$. As standard GNNs [11, 18] are computationally prohibitive on large graphs, some research works [4, 19] suggest decoupling the non-linear transformation from the feature propagation. In addition, LightGCN [12] empirically reveals that non-linear transformation operation will unnecessarily complicate the model and even degrade the effectiveness of recommendation models. Therefore, we follow these approaches and calculate the disentangled feature propagation in the pre-computation process.

Recent studies on node representation learning [7, 52, 55, 61] have shown that PPR plays an important role in capturing graph information. Based on this observation, recent studies [4, 38] propose approximate algorithms for feature propagation. However, to calculate the approximate PPR values by adopting existing algorithms [14, 15, 20, 24, 39–43, 47], PPRGo [4] still requires linear costs for

Algorithm 1: Global-Feature-Propagation

Input: Graph $\mathcal{G}$, feature $\boldsymbol{X}$, threshold ϵ, stopping probability α
Output: Representation $\boldsymbol{H}$

1 Initialize $\boldsymbol{H} \leftarrow \boldsymbol{0}$, $\boldsymbol{X} \leftarrow$ Column-normalized$(\boldsymbol{X})$
2 **for** $k \in \{1, ..., F+1\}$ **do**
3 **while** $\exists u \in \mathcal{U} \cup C$ *such that* $\boldsymbol{X}_{u,k} > \epsilon \cdot d_u$ **do**
4 $\boldsymbol{H}_{u,k} \leftarrow \boldsymbol{H}_{u,k} + \alpha \cdot \boldsymbol{X}_{u,k}$
5 **for** $v \in \mathcal{N}(u)$ **do**
6 $\boldsymbol{X}_{v,k} \leftarrow \boldsymbol{X}_{v,k} + (1-\alpha) \cdot \frac{\boldsymbol{X}_{u,k}}{d_u}$
7 $\boldsymbol{X}_{u,k} \leftarrow 0$
8 **return** $\boldsymbol{H}$

feature propagation, which is still too expensive for large graphs. AGP [38] assumes that the given feature vector $\boldsymbol{x}$ is L_1-normalized, i.e., $\sum_{i=1}^{n+n_c} \boldsymbol{x}_i = 1$, where each dimension corresponds to the feature value of a node in the graph. Then, the running cost can be bounded by $O(F/\epsilon)$, which is independent of the graph size. However, in many cases, such limited propagation is insufficient. To tackle this issue, we make a trade-off by incorporating a column-wised feature propagation with L_2 normalization, which runs in $O(F\sqrt{n+n_c}/\epsilon)$ time and gains a balance between efficiency and effectiveness.

Algorithm 1 shows the pseudo-code of the global feature propagation algorithm. Given a column of features $\boldsymbol{X}_{:,k}$, the output representation vector $\boldsymbol{H}_{:,k}$ is maintained to store the portion of feature information propagated to each node via α-discount random walks, where at each propagation step, the current feature information either stores at the current node with α probability or randomly propagates to an out-neighbor of the current node with $(1-\alpha)$ probability. Besides, the vector $\boldsymbol{X}_{:,k}$ indicates the portion of feature information currently propagated at each node but has not been stored yet. Thus, if the entries of $\boldsymbol{X}_{:,k}$ are all zero values, it returns the exact propagation results.

Initially, the entries of $\boldsymbol{H}_{:,k}$ are all zero (Line 1), indicating that the feature information of each node has not been stored yet. We further normalize the feature matrix before the propagation process. Then, the features are propagated separately along each dimension (Line 2). For the k-th column of the feature matrix, if any entry $\boldsymbol{X}_{u,k}$ of the

feature matrix is above $\epsilon \cdot d_u$ (Line 3), where ϵ is a given threshold and $d_u = |\mathcal{N}(u)|$ is the out-degree of node u, a push operation is invoked on u to propagate features to its neighbors via α-discount random walks (Lines 4-7). In particular, it first converts $\alpha \cdot \boldsymbol{X}_{u,k}$ to $\boldsymbol{H}_{u,k}$ (Line 4). To explain, α portion of the feature information $\boldsymbol{X}_{u,k}$ stores at the current node u. Next, the remaining $(1-\alpha)$ portion of the feature information $\boldsymbol{X}_{u,k}$ will evenly propagate to the out-neighbors of the current node u (Lines 5-6). Thus, for each v that is an out-neighbor of u, the feature $\boldsymbol{H}_{v,k}$ is increased by $(1-\alpha) \cdot \boldsymbol{X}_{u,v}/d_u$. After the push operation, the feature information $\boldsymbol{X}_{u,k}$ is set to zero (Line 7), since $\boldsymbol{X}_{u,k}$ has been either propagated to neighbors or stores at node u. This algorithm will terminate when there exists no node u such that $\boldsymbol{X}_{u,k}$ is larger than $\epsilon \cdot d_u$. The following theorem shows that with the global propagation algorithm, the representations $\boldsymbol{H}$ can be generated in sub-linear time, and the proof can be found in Appendix.

Theorem 2. *Generating the representation matrix $\boldsymbol{H}$ using Algorithm 1 takes $O(F\sqrt{n+n_c}/\epsilon)$ time complexity given that the original feature matrix $\boldsymbol{X}$ is L_2-normalized by each column.*

3.4 Local Feature Propagation

Different from user-item recommendation tasks, the community nodes and their members constitute explicit subgraphs, providing additional local structural information. Furthermore, the positive training data is extremely sparse due to the user constraint. To generate more informative representations of communities, we present the local computation component to better capture users' preferences in the same community. The local feature propagation of ComRec consists of two steps: local update and subgraph propagation.

Local update. Pooling functions are common methods in many graph tasks, such as graph classification [57, 58]. The embeddings of nodes within the graph are combined together to generate the graph representation:

$$\boldsymbol{H}_c = \text{pooling}(\{\boldsymbol{H}_u | \forall u \in c\}),$$

where the pooling functions can be *min*, *max*, and *average* functions. In ComRec, we extend this idea to generate the representations of the community nodes. Thus, the representation of the community node c is as follows:

$$\boldsymbol{H}_c = \boldsymbol{H}_c + \sum_{u \in c} \alpha_u \boldsymbol{H}_u,$$

where α_u is an individualized preference weight of user u in the community c and can be either calculated by attention mechanism with trainable weights or set to be a hyper-parameter. In our implementation, we find that uniform aggregation is enough to achieve satisfactory recommendation results, and thus we set $\alpha_u = 1$, which also avoids weight updating costs. With the local update step, the representation of the community node will capture more member preference information, which can also be regarded as the preference summarization of this community.

Subgraph propagation. The local update process is designed to incorporate user preference in the community representation while it does not capture the close relationships between in-community members. Since the explicit subgraphs that consist of members and communities provide additional local structural information, to make full use of such information, we further propagate node representations locally, which can smooth the node representations within each community. Specifically, we adopt the normalized sum aggregator in each community and remove the non-linear transformation as [12]. The k-th layer representation of node $i \in c$ during the local propagation is defined as:

$$\boldsymbol{H}_i = \sum_{j \in \mathcal{N}_c(i)} \frac{1}{\sqrt{|\mathcal{N}_c(i)|}\sqrt{|\mathcal{N}_c(j)|}} \boldsymbol{H}_j,$$

where $\mathcal{N}_c(i)$ is the neighbor set of node i in the subgraph generated by community c. Notice that no self-loop is considered during the subgraph propagation process since α-discounted random walks have been utilized during the global feature propagation.

3.5 Training and Prediction

After two propagation steps, we apply a *multi-layer perceptron (MLP)* to get the final embeddings of each node, i.e., $\boldsymbol{Z} = \text{MLP}(\boldsymbol{H})$. The final prediction score is defined as the inner product of the user and the community embeddings:

$$\hat{r}_{uc} = \boldsymbol{Z}_u^T \boldsymbol{Z}_c. \tag{1}$$

The communities with top-K scores will be returned as the final recommendation results for users.

Loss function. In our ComRec, the trainable parameters are only the weight matrices of MLP. Therefore, it supports efficient mini-batch training and scale to large graphs. We select the standard Bayesian Personalized Ranking loss [28] to maximize the differences between positive pairs and negative pairs:

$$\mathcal{L} = -\sum_{u \in \mathcal{U} \cup C} \log\left(\sigma(\hat{r}_{uc} - \hat{r}_{uc'})\right) + ||\boldsymbol{W}||_2^2,$$

where c is the positive sample and c' is the negative sample, σ is the sigmoid function, and $\boldsymbol{W}$ denotes all trainable parameters.

Training samples. The positive samples are the observed affiliations of users. For the negative samples, if we directly sample a number of random communities for each user, the chance of any of these communities being related to the user is small. Therefore, we adopt the hard negative sampling strategy [27, 56] to get more relative negative samples for each user. Specifically, for each user u, we calculate its single source PPR values and select the communities with the top-5 PPR scores as hard negative samples.

Candidate selection. Since each user can only join at most one community, given a certain user u, most of the communities are irrelative to u and unlikely to be valid for the recommendation. To make the prediction more reasonable, we first select a candidate set for each user u according to the game logs and user profile, such as the communities that other players in the same game have joined, and the communities that a friend of u has joined. After the candidate selection, we only recommend communities in the candidate set of each user via Equation 1.

4 EXPERIMENTS

We compare our ComRec against alternative solutions on the constrained social community recommendation task.

Table 2: Dataset statistics.

Game	Players	Clubs	Edges	Features
X-1	5,018,992	116,080	24,802,796	38
X-2	4,868,207	120,805	18,422,695	38
Y	5,652,790	77,953	80,581,339	28
Z	1,722,069	59,545	12,181,427	26

Table 3: Experimental results on X-1.

Dataset	X-1		
Method	Hit@10	NDCG@10	Total Time(s)
DistNE	0.50623	0.26183	3440
MLP	0.48365	0.24273	**1037**
LightGCN	0.49239	0.25589	1431
LightGCN-e	0.53181	0.28496	1711
GraphRec	0.51006	0.25596	80604
AutointCL	0.50994	0.25546	26352
AttentionNet	0.52023	0.27782	143605
ComRec	**0.56343**	**0.30157**	1239

Table 4: Experimental results on X-2.

Dataset	X-2		
Method	Hit@10	NDCG@10	Total Time(s)
DistNE	0.51093	0.24610	3525
MLP	0.48097	0.23563	**1246**
LightGCN	0.51097	0.25325	1544
LightGCN-e	0.52442	0.26334	1802
GraphRec	0.49843	0.23498	83043
AutointCL	0.49982	0.23515	30817
AttentionNet	0.51093	0.24594	115910
ComRec	**0.55544**	**0.29705**	1337

4.1 Experimental Settings

Datasets. Since no public dataset considers the constraint in our setting, in this paper, we evaluate our ComRec and several competitors on three Tencent mobile *strategy role-playing game (SRPG)* datasets, denoted by X, Y, and Z, respectively. Besides, the dataset from game X is further divided into X-1 and X-2 according to their mobile platforms. For graphs X-1 and X-2, we construct *(i)* the social graphs by taking each *daily active user (DAU)* in the game as a user node and the friendship between two users as an edge; *(ii)* the user-community graph by taking the club that at least one DAU has joined as a community node, the DAU in the club as a user node, and the memberships between the user nodes and the club nodes as edges. For graph Y and graph Z, we construct *(i)* the social graphs by all users and their friendships; *(ii)* the bipartite graphs by all users, clubs, and their memberships. The statistics of these datasets are listed in Table 2.

Competitors. We evaluate ComRec against 7 competitors, including embedding methods, collaborative filtering methods, contrastive learning methods, and social recommendation methods. We obtain the source code of these competitors from GitHub and evaluate them with default parameter settings suggested by their authors unless otherwise specified. We list these methods as follows:

- DistNE [21]: it is a distributed embedding method based on a graph partitioning algorithm [22]. The embeddings are computed by node2vec [10]. Then, the embeddings are fed into a multi-layer perceptron as features to obtain the pair-wise scores;
- MLP: it directly adopts a multi-layer perceptron to train embeddings using features;
- AutoIntCL: it combines the attention-based recommendation approach autoint[2] [31] with the contrastive learning loss function, aiming to maximize the distances between negative pairs and minimize the distances between positive pairs. It has been deployed in real Tencent applications;
- AttentionNet: it is a deep model [16] with the multi-head self-attention mechanism [37]. It focuses more on the important features and reduces the impact of irrelevant features. It has been deployed in real Tencent applications.
- LightGCN[3] [12]: it is one of the state-of-the-art neural collaborative filtering approaches that can scale to large graphs with millions of nodes;
- LightGCN-E: it denotes an extended LightGCN model that further considers the social network as input;
- GraphRec[4] [8]: it is one of the state-of-the-art social recommendation approaches which takes the extended graph as input and generated embeddings by attention-based message-passing.

Table 5: Experimental results on Y.

Dataset	Y		
Method	Hit@5	NDCG@5	Total Time(s)
DistNE	0.21342	0.1262	5724
MLP	0.19430	0.11288	**1684**
LightGCN	0.23763	0.1384	1724
LightGCN-e	0.20289	0.11466	2156
GraphRec	0.19353	0.11013	125376
AutointCL	0.19509	0.12718	53696
AttentionNet	0.21431	0.13661	216888
ComRec	**0.25049**	**0.14712**	2254

Table 6: Experimental results on Z.

Dataset	Z		
Method	Hit@5	NDCG@5	Total Time(s)
DistNE	0.49001	0.33243	1449
MLP	0.49499	0.31126	**470**
LightGCN	0.47043	0.32158	688
LightGCN-e	0.46796	0.32167	890
GraphRec	0.43252	0.3006	68548
AutointCL	0.39468	0.29250	15232
AttentionNet	0.38537	0.28973	29193
ComRec	**0.52805**	**0.34412**	887

Table 7: The percentage of friends in the same club.

Graph	X-1	X-2	Y	Z
Portion	0.59%	0.82%	0.34%	0.50%

Parameter settings. We set $\alpha = 0.1$, $\epsilon = 10^{-6}$ (introduced in Section 3.3). The depth of the MLP is 3 and the size of the hidden layer is 64. We select 5 hard negative samples for each user existing in positive samples. Since the restrictions for joining clubs in game Y and game Z are more strict than that in game X, we set the

[2]https://github.com/DeepGraphLearning/RecommenderSystems/tree/master/featureRec
[3]https://github.com/kuandeng/LightGCN
[4]https://github.com/wenqifan03/GraphRec-WWW19

Table 8: Ablation study on X-1.

Method	Hit@10	Δ	NDCG@10	Δ
ComRec-lm	0.51003	-9.48%	0.27941	-7.35%
ComRec-lu	0.48018	-14.78%	0.28960	-3.97%
ComRec-sp	0.54280	-3.66%	0.28413	-5.78%
ComRec	**0.56343**	-	**0.30157**	-

Table 9: Ablation study on Z.

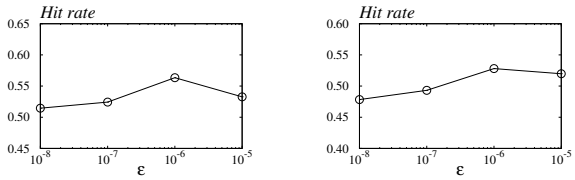
Method	Hit@5	Δ	NDCG@5	Δ
ComRec-lm	0.50793	-3.81%	0.32653	-5.11%
ComRec-lu	0.50159	-5.01%	0.33922	-1.42%
ComRec-sp	0.50675	-4.03%	0.33519	-2.59%
ComRec	**0.52805**	-	**0.34412**	-

candidate set sizes of graphs X-1, X-2, Y, and Z to be 100, 100, 50, and 30, respectively.

4.2 Offline Evaluation

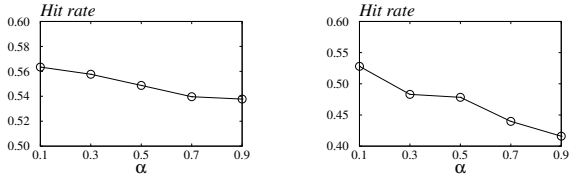
We first conduct experiments on the above-mentioned four datasets. We randomly hide 10% of the user-community edges for testing and train the embedding vectors on the rest of the graph, i.e., the whole user-user social graph and 90% of the user-community graph. For each user u, the candidate set is generated by the communities according to the following priority: *(i)* the ground-truth community that user u has joined; *(ii)* the communities that other players in the same game with user u have joined; *(iii)* the communities that at least one friend of user u has joined; *(iv)* randomly selected communities. Once the number of candidates equals the default number, the candidate selection process will terminate. Given a node pair (u, c) in the candidate set, we compute a score for (u, c) based on the embedding vectors generated by each model, and return the top-K results to evaluate model performances according to the predicted scores. Since the sizes of the candidate set in game X, game Y, and game Z are distinct, we set $K = 10, 10, 5, 5$ on graphs X-1, X-2, Y, and Z, respectively.

Table 3 and Table 4 show the experimental results and the total running time (including training and prediction) of all methods on two graphs of game X. We make the following observations. Firstly, on graphs X-1 and X-2, LightGCN-e consistently outperforms LightGCN in terms of Hit@10 and NDCG@10 metrics. This demonstrates that directly applying the collaborative filtering approach on the user-community bipartite graph may lose potential information in the social network. LightGCN-e performs information propagation on the extended graph and thus alleviates the training data sparse issue, which shows the effectiveness of incorporating social relationships between users into model training. Secondly, since the training data is extremely sparse compared with traditional recommendation tasks, GNN models with complex attention computation may be over-parameterized and thus overfits the training data, losing the generalization ability. Thirdly, our ComRec consistently outperforms other competitors on graph X-1 and graph X-2 by a large margin. Specifically, ComRec takes the lead by at least 5.95% and 5.92% in terms of hit rate on graphs X-1 and X-2, respectively; at least 5.83% and 12.80% in terms of NDCG on graphs X-1 and X-2, respectively. Fourthly, attention-based approaches, GraphRec, AutointCL, and transferCL, take more than an order of magnitude time than other approaches, demonstrating that the complex attention mechanism may not be appropriate for large graphs with millions of nodes. Compared with the second-best method, LightGCN-e, our ComRec saves 27.59% and 25.80% time on graphs X-1 and X-2, respectively. In addition, the global propagation process of ComRec only takes 6.7s and 6.2s on graphs X-1 and X-2, respectively.



(a) Graph X-1 (b) Graph Z

Figure 4: Varying ϵ on graph X-1 and Z.



(a) Graph X-1 (b) Graph Z

Figure 5: Varying α on graph X-1 and Z.

Table 5 and Table 6 show the experimental results of all methods on two graphs of games Y and Z. LightGCN outperforms LightGCN-e on these two graphs. Table 7 further summarizes the portion of friend pairs in the same club among all friend edges. As we can observe, the values on graphs Y and Z are much lower than that on graphs X-1 and X-2. Therefore, we conclude that graphs Y and Z show more heterogeneous properties and only propagating features on the extended graph is not enough to obtain satisfactory results on such graphs. Our ComRec further incorporates local structural information into representations and consistently outperforms other competitors on graphs Y and Z by up to 7.70% and 6.68%, respectively, in comparable running time. These results demonstrate the effectiveness and efficiency of our ComRec.

Ablation study. Next, we show the importance of each component in our ComRec. We use ComRec-lm to indicate the algorithm that removes the labeling mechanism component, ComRec-lu to indicate the algorithm that removes the local update component, and ComRec-sp to indicate the algorithm that removes the subgraph propagation component. Table 8 and Table 9 list the results of these approaches on graphs X-1 and Z, respectively. As we can observe, ComRec-lm, ComRec-lu, and ComRec-sp show sub-optimal results on X-1 and Z in terms of both hit rate and ndcg metrics, while our ComRec achieves the best performance. Moreover, ComRec benefits from the local update component most in terms of hit rate and benefits from the labeling mechanism component most in terms of ndcg metric. These results demonstrate the effectiveness of each component in our framework.

Parameter analysis. We conduct experiments to analyze the effect of parameters α and ϵ on graphs X-1 and Z in terms of hit rate, results on other graphs show similar trends. Figure 4 shows the recommendation results as we vary ϵ from 10^{-8} to 10^{-5}. As ϵ approaches 10^{-8}, we obtain more accurate representations; as ϵ approaches 10^{-5}, less information will be propagated in the graph. We observe that when $\epsilon = 10^{-6}$, ComRec achieves the best results and thus ϵ is set to be 10^{-6} in our experiments. Figure 5 shows

Table 10: The average statistics of the graphs in game X.

Game	Type	Players	Clubs	Edges
X	SRPG	20.95million	0.28million	0.16billion

the recommendation results as we vary α from 0.1 to 0.9. As α approaches 0.1, multi-hop information will be incorporated into the representations; as α approaches 0.9, it will focus more on one-hop neighbors during the feature propagation, which results in performance degradation. Hence, we set $\alpha = 0.1$ in our experiments.

5 DEPLOYMENT

We deploy the proposed ComRec in Tencent mobile game X with a real application that recommends in-game clubs for each player.

5.1 Setup

Firstly, we select active users that have game records in the past two weeks according to their game logs. Then, we construct *(i)* the social graph by taking active users as nodes and the friendships between them as user-user edges; *(ii)* the bipartite graph by taking active users and existing clubs as nodes, the memberships between users and clubs as user-community edges. Besides, we update the recommendation results of players every day by re-running the algorithms from scratch on the latest dataset and report the results over a consecutive period of 7 days. Table 10 shows the average statistics of the graphs in game X. We compare ComRec against several alternative approaches that have been deployed for this recommendation application as follows:

- Random: it randomly recommends clubs for each player;
- AutointCL: it adopts the same algorithm as in Section 4.2.
- AutointCL-M: it applies distributed AutointCL algorithm on the candidate set generated by a multi-retrieval approach, i.e., its candidate sets are different from that of AutointCL.
- AttentionNet: it adopts the same algorithm as in Section 4.2.

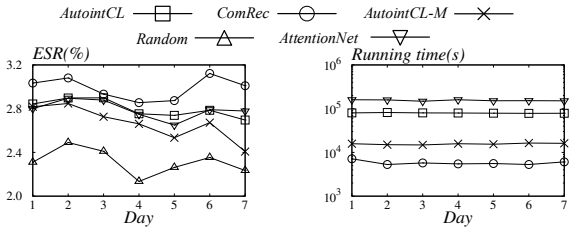
For each method, the candidate set of each user consists of the clubs that other players in the same game have joined according to the game logs except for AutointCL-M, which considers multiple types of clubs to generate the candidate sets. After that, each method will generate a recommendation list for each user following the same setting introduced in Section 4.1. These approaches are evaluated by the online A/B testing [17, 33] that randomly assigns each approach to a fraction of users, i.e., the players receive the recommendations from different approaches.

5.2 Online Evaluation

We provide an in-game module to recommend an ordered list of clubs to each player. When the player u accesses the club recommendation module in game X, u will see one recommended club each time. This will generate an *exposure* record in recommendation logs and u can decide to click or not. If u is not interested in the current club, u can switch to the next recommended result. Once user u clicks the recommended club, it will send a join request and generate a *click* record in recommendation logs. If the request needs approval, the administrators of the club can decide to accept the request or reject it. Besides, the user u needs to satisfy the role level and rank level requirements of the club c if u wants to join c.

Table 11: Average pair-wise CR and overall ESR on game X.

Method	CR	ESR
Random	0.03228	0.04930
AutointCL	0.03774	0.05469
AutointCL-M	0.03940	0.05493
AttentionNet	0.03806	0.05493
ComRec	**0.04030**	**0.05736**



(a) Pair-wise ESR (b) Running time

Figure 6: Pair-wise ESR and running time on game X.

We evaluate ComRec and other competitors for club recommendation by three metrics: *(i)* pair-wise *click rate (CR)*, the fraction of recommended records that are clicked by users; *(ii)* pair-wise *exposure success rate (ESR)*, the fraction of recommended records that users successfully join the club; *(iii)* overall ESR, the fraction of users that successfully join one recommended club.

Table 11 shows the average pair-wise CR and overall ESR of each method in a consecutive period of 7 days on game X. Firstly, our ComRec takes the lead by at least 2.28% compared with other approaches in terms of average pair-wise CR, which shows that players are more interested in the clubs recommended by ComRec. Secondly, our ComRec takes the lead by 4.40% compared with the second-best approach in terms of overall ESR, demonstrating that ComRec can generate recommendation lists of higher quality than other approaches. Figure 6 illustrates the pair-wise ESR and running time of each method in a consecutive period of 7 days. As we can observe, our ComRec consistently outperforms other competitors every single day. In particular, our ComRec takes the lead by at least 6.61% and up to 29.09% in terms of the average pair-wise ESR. Besides, compared with AutointCL and AttentionNet, our ComRec significantly reduces the running time by at least an order of magnitude. These results again demonstrate the effectiveness and efficiency of ComRec.

6 CONCLUSION

In this paper, we focus on the social community recommendation problem with additional constraint in real-world applications. To solve this problem, we present ComRec, an effective and efficient recommendation framework by capturing both global and local information. Extensive experiments demonstrate the effectiveness and efficiency of our ComRec.

ACKNOWLEDGMENTS

This work is supported by the Hong Kong RGC ECS grant (No. 24203419), RGC GRF grant (No. 14217322), RGC CRF grant (No. C4158-20G), Hong Kong ITC ITF grant (No. MRP/071/20X), NSFC grant (No. U1936205), and CUHK Direct Grant (No. 4055181).

REFERENCES

[1] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. 2020. Subgraph neural networks. In *NeurIPS*. 8017–8029.

[2] Aris Anagnostopoulos, Ravi Kumar, and Mohammad Mahdian. 2008. Influence and correlation in social networks. In *SIGKDD*. 7–15.

[3] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. 2006. Group Formation in Large Social Networks: Membership, Growth, and Evolution. In *SIGKDD*. 44–54.

[4] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *SIGKDD*. 2464–2473.

[5] Da Cao, Xiangnan He, Lianhai Miao, Yahui An, Chao Yang, and Richang Hong. 2018. Attentive Group Recommendation. In *SIGIR*. 645–654.

[6] Wen-Yen Chen, Dong Zhang, and Edward Y. Chang. 2008. Combinational Collaborative Filtering for Personalized Community Recommendation. In *SIGKDD*. 115–123.

[7] Xinyu Du, Xingyi Zhang, Sibo Wang, and ZengFeng Huang. 2023. Efficient Tree-SVD for Subset Node Embedding over Large Dynamic Graphs. *PACMMOD* 1, 1 (2023), 96:1–96:26.

[8] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *WWW*. 417–426.

[9] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*. 1263–1272.

[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. 855–864.

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[12] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, YongDong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. 639–648.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.

[14] Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. 2021. Massively parallel algorithms for personalized pagerank. *PVLDB* 14, 9 (2021), 1668–1680.

[15] Guanhao Hou, Qintian Guo, Fangyuan Zhang, Sibo Wang, and Zhewei Wei. 2023. Personalized PageRank on Evolving Graphs with an Incremental Index-Update Scheme. *PACMMOD* 1, 1 (2023), 25:1–25:26.

[16] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *CVPR*. 4700–4708.

[17] Shixun Huang, Wenqing Lin, Zhifeng Bao, and Jiachen Sun. 2022. Influence Maximization in Real-World Closed Social Networks. *PVLDB* 16, 2 (2022), 180—-192.

[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[19] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.

[20] Wenqing Lin. 2019. Distributed algorithms for fully personalized pagerank on large graphs. In *WWW*. 1084–1094.

[21] Wenqing Lin. 2021. Large-Scale Network Embedding in Apache Spark. In *SIGKDD*. 3271–3279.

[22] Wenqing Lin, Feng He, Faqiang Zhang, Xu Cheng, and Hongyun Cai. 2020. Initialization for Network Embedding: A Graph Partition Approach. In *WSDM*. 367–374.

[23] Zhiwei Liu, Liangwei Yang, Ziwei Fan, Hao Peng, and Philip S Yu. 2022. Federated social recommendation with graph neural network. *TIST* 13, 4 (2022), 1–24.

[24] Siqiang Luo, Xiaokui Xiao, Wenqing Lin, and Ben Kao. 2019. Efficient Batch One-Hop Personalized PageRanks. In *ICDE*. 1562–1565.

[25] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* (2001), 415–444.

[26] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the web. (1999).

[27] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *WSDM*. 273–282.

[28] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*. 452–461.

[29] Aravind Sankar, Yanhong Wu, Yuhang Wu, Wei Zhang, Hao Yang, and Hari Sundaram. 2020. Groupim: A mutual information maximization framework for neural group recommendation. In *SIGIR*. 1279–1288.

[30] Amit Sharma and Baoshi Yan. 2013. Pairwise Learning in Recommendation: Experiments with Community Recommendation on Linkedin. In *RecSys*. 193–200.

[31] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *CIKM*. 1161–1170.

[32] Ellen Spertus, Mehran Sahami, and Orkut Buyukkokten. 2005. Evaluating similarity measures: a large-scale study in the orkut social network. In *SIGKDD*. 678–684.

[33] Diane Tang, Ashish Agarwal, Deirdre O'Brien, and Mike Meyer. 2010. Overlapping experiment infrastructure: More, better, faster experimentation. In *SIGKDD*. 17–26.

[34] Jiliang Tang, Charu Aggarwal, and Huan Liu. 2016. Recommendations in signed social networks. In *WWW*. 31–40.

[35] Jiliang Tang, Xia Hu, and Huan Liu. 2013. Social recommendation: a review. *SNAM* 3, 4 (2013), 1113–1133.

[36] Vishvas Vasuki, Nagarajan Natarajan, Zhengdong Lu, and Inderjit S. Dhillon. 2010. Affiliation Recommendation Using Auxiliary Networks. In *RecSys*. 103–110.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *NeurIPS*. 6000–6010.

[38] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2021. Approximate graph propagation. In *SIGKDD*. 1686–1696.

[39] Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized pagerank to a target node, revisited. In *SIGKDD*. 657–667.

[40] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. Hubppr: effective indexing for approximate personalized pagerank. *PVLDB* 10, 3 (2016), 205–216.

[41] Sibo Wang and Yufei Tao. 2018. Efficient algorithms for finding approximate heavy hitters in personalized pageranks. In *SIGMOD*. 1113–1127.

[42] Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. 2019. Efficient algorithms for approximate single-source personalized pagerank queries. *TODS* 44, 4 (2019), 1–37.

[43] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *SIGKDD*. 505–514.

[44] Xin Wang, Roger Donaldson, Christopher Nell, Peter Gorniak, Martin Ester, and Jiajun Bu. 2016. Recommending Groups to Users Using User-Group Engagement and Time-Dependent Matrix Factorization. In *AAAI*. 1331–1337.

[45] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *SIGIR*. 165–174.

[46] Xiyuan Wang and Muhan Zhang. 2022. GLASS: GNN with Labeling Tricks for Subgraph Representation Learning. In *ICLR*.

[47] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. In *SIGMOD*. 441–456.

[48] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *SIGIR*. 726–735.

[49] Le Wu, Junwei Li, Peijie Sun, Richang Hong, Yong Ge, and Meng Wang. 2020. Diffnet++: A neural influence and interest diffusion network for social recommendation. *TKDE* (2020).

[50] Le Wu, Peijie Sun, Yanjie Fu, Richang Hong, Xiting Wang, and Meng Wang. 2019. A neural influence diffusion model for social recommendation. In *SIGIR*. 235–244.

[51] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks. In *ICLR*.

[52] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. 2020. Homogeneous Network Embedding for Massive Graphs via Reweighted Personalized PageRank. *PVLDB* 13, 5 (2020), 670–683.

[53] Hongzhi Yin, Qinyong Wang, Kai Zheng, Zhixu Li, Jiali Yang, and Xiaofang Zhou. 2019. Social influence-based group representation learning for group recommendation. In *ICDE*. 566–577.

[54] Hongzhi Yin, Qinyong Wang, Kai Zheng, Zhixu Li, Jiali Yang, and Xiaofang Zhou. 2019. Social influence-based group representation learning for group recommendation. In *ICDE*. 566–577.

[55] Yuan Yin and Zhewei Wei. 2019. Scalable Graph Embeddings via Sparse Transpose Proximities. In *SIGKDD*. 1429–1437.

[56] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *SIGKDD*. 974–983.

[57] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. *NeurIPS* 31 (2018).

[58] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *AAAI*, Vol. 32.

[59] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. 2021. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *NeurIPS* 34 (2021), 9061–9073.

[60] Shiqi Zhang, Jiachen Sun, Wenqing Lin, Xiaokui Xiao, and Bo Tang. 2022. Measuring Friendship Closeness: A Perspective of Social Identity Theory. In *CIKM*. 3664–3673.

[61] Xingyi Zhang, Kun Xie, Sibo Wang, and Zengfeng Huang. 2021. Learning Based Proximity Matrix Factorization for Node Embedding. In *SIGKDD*. 2243–2253.

A PROOF OF THEOREM

Proof of Theorem 2. Let $\boldsymbol{X}^{(l_2)}$ denote the original feature matrix after taking column-wise L_2 normalization. According to the definitions, $\boldsymbol{X}_{:,k}$ indicates the portion of feature information currently propagated at each node but has not been stored yet, $\boldsymbol{H}_{:,k}$ indicates the portion of feature information stored at each node. For the k-th column $\boldsymbol{X}^{(l_2)}_{:,k}$, we divide its entries by $L_1 = \left\|\boldsymbol{X}^{(l_2)}_{:,k}\right\|_1$, which is a constant determined by the original feature values. So we have $\sum_{u\in\mathcal{U}\cup C} \boldsymbol{X}^{(l_2)}_{u,k}/L_1 = 1$. Then, the following invariant always holds for each column k:

$$\left\|\frac{\boldsymbol{X}^{(l_2)}_{:,k}}{L_1}\right\|_1 = \left\|\frac{\boldsymbol{H}_{:,k}}{L_1}\right\|_1 + \left\|\frac{\boldsymbol{X}_{:,k}}{L_1}\right\|_1.$$

In particular, the initial states satisfy the above equation, by induction, it can be proved that the above equation always holds after each push operation.

Wang et al. [43] have shown that forward push algorithms can be extended to arbitrary source distributions, in which the PPR values can still be computed without compromising their asymptotic guarantees. Since $\sum_{u\in\mathcal{U}\cup C} \boldsymbol{X}^{(l_2)}_{u,k}/L_1 = 1$, $\boldsymbol{X}^{(l_2)}_{:,k}/L_1$ can be regarded as a distribution generated by the k-th feature vector. Recap from Algorithm 1 that, it propagates features using α-discount random walks. When $\left\|\boldsymbol{X}_{:,k}\right\|_1 = 0$, the exact propagation results will be stored in $\boldsymbol{H}_{:,k}$. Therefore, calculating $\boldsymbol{H}_{:,k}/L_1$ is equivalent to calculating PPR values on the distribution initialized by $\boldsymbol{X}^{(l_2)}_{:,k}/L_1$. When $\left\|\boldsymbol{X}_{:,k}/L_1\right\|_1 = 0$, we can derive the true PPR values corresponding to the distribution initialized by $\boldsymbol{X}^{(l_2)}_{:,k}/L_1$, i.e., $\left\|\boldsymbol{X}^{(l_2)}_{:,k}/L_1\right\|_1 = \left\|\boldsymbol{H}_{:,k}/L_1\right\|_1$; otherwise, $\left\|\boldsymbol{H}_{:,k}/L_1\right\|_1 \le \left\|\boldsymbol{X}^{(l_2)}_{:,k}/L_1\right\|_1$.

The cost of the feature propagation is dominated by the number of push operations. Given a node u on feature dimension k, the push operation is invoked on node u if and only if $\boldsymbol{X}_{u,k} > \epsilon \cdot d_u$. Let $\boldsymbol{H}^l_{u,k}$ denote the stored value that is propagated to node u with l-hop random walks. Then the total number of push operations caused by the stored value $\boldsymbol{H}^l_{u,k}$ can be bounded by $\frac{\boldsymbol{H}^l_{u,k}}{\alpha\cdot\epsilon\cdot d_u}$. In addition, the cost of each push operation on node u is $O(d_u)$. Therefore, the total cost of the push operations caused by $\boldsymbol{H}^l_{u,k}$ is bounded by $\frac{\boldsymbol{H}^l_{u,k}}{\alpha\cdot\epsilon\cdot d_u}\cdot d_u$. The total time cost of Algorithm 1 on all nodes is:

$$\begin{aligned}&\sum_{k=1}^{F+1}\sum_{l=0}^{\infty}\sum_{u\in\mathcal{U}\cup C}\frac{\boldsymbol{H}^l_{u,k}}{\alpha\cdot\epsilon\cdot d_u}\cdot d_u\\ \le&\frac{1}{\alpha\cdot\epsilon}\sum_{k=1}^{F+1}\sum_{u\in\mathcal{U}\cup C}\sum_{l=0}^{\infty}\boldsymbol{H}^l_{u,k} \le \frac{1}{\alpha\cdot\epsilon}\sum_{k=1}^{F+1}\sum_{u\in\mathcal{U}\cup C}\boldsymbol{X}^{(l_2)}_{u,k},\end{aligned}$$

Since $\boldsymbol{X}^{(l_2)}$ is L_2-normalized, we have $\sum_{u\in\mathcal{U}\cup C}\left(\boldsymbol{X}^{(l_2)}_{u,k}\right)^2 = 1$. Then, by Cauchy–Schwarz inequality, we have

$$\sum_{u\in\mathcal{U}\cup C}\boldsymbol{X}^{(l_2)}_{u,k} \le \sqrt{n+n_c}.$$

Therefore, we can derive that

$$\frac{1}{\alpha\cdot\epsilon}\sum_{k=1}^{F+1}\sum_{u\in\mathcal{U}\cup C}\boldsymbol{X}^{(l_2)}_{u,k} \le \frac{1}{\alpha\cdot\epsilon}\sum_{k=1}^{F+1}\sqrt{n+m} = \frac{F+1}{\alpha\cdot\epsilon}\cdot\sqrt{n+m},$$

where α can be treated as a constant number. Thus, the total time complexity of Algorithm 1 on all dimensions is bounded by $O(F\sqrt{n+m}/\epsilon)$. □